Existence dependency-based domain modeling for improving stateless process enactment

Raf Haesen[1,2], Monique Snoeck[1], Wilfried Lemahieu[1], and Stephan Poelmans[1,2]

[1]Department of Decision Sciences & Information Management,
Katholieke Universiteit Leuven, Belgium
FirstName.LastName@econ.kuleuven.be
[2]Faculty of Economics & Management,
Hogeschool-Universiteit Brussel, Belgium
FirstName.LastName@hubrussel.be

Abstract

In a process-enabled service oriented architecture, a process engine typically stores the state of the process instances during enactment. As an alternative, stateless process enactment entails that process state is derived from the state of business objects, which are organized in a domain model. The business objects are referred to in pre- and post-conditions of activities, which determine when the activity is enabled and completed, respectively. Despite the fact that the latter approach has multiple benefits compared with the former, the repeated state (re)calculations deteriorate performance and the formulation of clear conditions is not self-evident if typical domain modeling techniques (e.g. UML or ER) are adopted. In this paper we show that by adopting a specific domain modeling technique, which is based on the notion of existence dependency between the business objects, the performance and comprehensibility issues can proficiently be dealt with. We illustrate the technique using a real-world case from the insurance domain and analyze the emerging duality between process modeling and domain modeling.

1 Introduction

In a typical service oriented architecture (SOA), workflow services support the coordination of a set of activities according to a business process. Workflow services are typically delivered by a process engine, which stores the state of the process instances as part of its workflow control data [14]. The problem with this *stateful* approach is that synchronization problems may occur between the process state and the state of business objects that are manipulated during the execution of the activities. These state inconsistencies may arise if the process engine fails during the execution of an activity or when a human actor chooses to perform an activity outside control of the workflow engine [5]. In both cases, the process engine does not get informed of changes to business objects, so that process state is not getting updated appropriately.

In an attempt to tackle these issues, Haesen et al. [4] described a pattern for *stateless* process enactment, in which the state of a process is derived from the state of the business objects. More specifically, each activity has a set of pre- and post-conditions, which express facts about the existence and/or state of one or more (attributes of) business objects. If the pre-conditions of a particular activity in the context of a process are met, that activity can be executed. On the other hand, an activity is completed if all its post-conditions are met.

In order verify the pre- and post-conditions, some information about the business objects must be retrieved. Therefore, the pattern assumes the existence of a particular domain model according to which the business object types are organized. However, no attention was paid to the requirements of such a model. As such, any common domain modeling technique, such as UML class diagrams or ER diagrams, would have been appropriate to define the domain model.

Upon closer inspection, it appears that the adoption of

these ‘classical’ domain modeling techniques suffer from some drawbacks: we will illustrate that the weak coherence between the business objects (a) prevents the formulation of comprehensible pre- and post-conditions and (b) causes performance issues. On the other hand, these issues can be alleviated when another, more specific domain modeling technique is adopted. This technique, which is based on the notion of existence dependency between business objects, prescribes how to create a so-called *existence dependency graph* (EDG) as domain model.

The remainder of the paper is organized as follows. Section 2 summarizes the principles of stateless process enactment, some of its advantages, but especially its weaknesses when UML is used as domain modeling language. Section 3 will explain existence dependency and its role in the construction of an EDG. In section 4, stateless process design and enactment is revised using an EDG as domain model. The new modeling and enactment approaches are discussed and evaluated using a real-life case from the insurance services domain. In section 5, we compare our work with related work and finally, section 6 presents the conclusions and some areas of future work.

2. Stateless process enactment

A process model for stateless enactment consists of a set of activities, whereby each activity has a set of pre- and post-conditions. It should be noted that such a set of activities actually presents a process model. Instead of explicitly representing the control-flow between the activities, constraints on the execution are imposed by means of the pre- and post-conditions. These conditions refer to business object types, which are organized according to a particular domain model.

As an example, consider the activity ‘Pay Indemnity’ of a claim handling process with its accompanying pre- and post-conditions. These conditions refer to the insurance domain model that is represented in Figure 1 (in order not to overload the model, attributes were omitted). The pre-conditions state that the activity can be executed if an INDEMNITY CALCULATION object exists, if the indemnity is determined, and if an ACCOUNT object exists. After the execution of the activity, an INDEMNITY PAYMENT object is created and its associations are established.

```
Pay Indemnity
 pre:
   INDEMNITY CALCULATION ic exists
   ic.amount filled out
   ACCOUNT a exists
 post:
   INDEMNITY PAYMENT ip exists
   ip.account = a
```

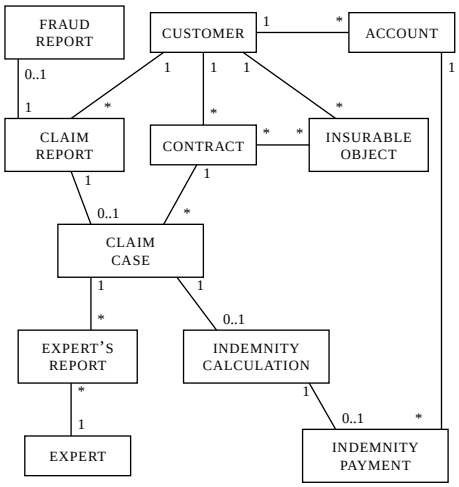


Figure 1. UML insurance domain model

```
ip.indemnity calculation = ic
```

Stateless process enactment clearly avoids state synchronization issues since the enabling and completion of activities is derived from the state of the business objects. Additionally, it offers many other benefits (known as positive forces in the pattern community), such as an increase in flexibility. Indeed, the pre-conditions that impose data dependencies that can be considered as the only (hard) constraints that must be fulfilled, so that much freedom is left at runtime. When desired, the modeler can add other (soft) constraints by formulating the appropriate pre- and post-conditions. Besides catering for more flexibility, executability is improved since the process engine can unambiguously determine when an activity is enabled and completed by validating the pre- and post-conditions, respectively [4].

On the other hand, this approach to stateless process enactment also suffers from some drawbacks. Firstly, some arbitrary navigation from one object to another is not possible since the existence of an object is not guaranteed. For example, the activity ‘Pay Indemnity’ was previously introduced, which transfers money to an account. However the ACCOUNT object is possibly not existing and therefore unreachable from CUSTOMER or INDEMNITY PAYMENT. Moreover, the many-to-many relationships anyhow prevent unique navigation between objects. For example, a contract specifies the insurance of multiple insurable objects and an insurable object can be insured in multiple contracts. Nevertheless, a claim case depends on a single contract and a single insurable object. Lastly, even if an object is existing and uniquely navigable, UML has no constructs to assure that the navigated object is *always* the same. For example, a claim case refers to a single contract that covers the claim. However, the semantics of the UML model do not prevent to unintendedly re-assign the claim case to another contract.

These reasons illustrate that references to *all* objects in the domain model must be maintained and state calculation requires the checking of *all* conditions. As such, the repeated (re)calculations of process state deteriorates performance, especially if the domain model consists of many business objects to be queried. Moreover, for the same reasons the formulation of clear and uniform conditions is not self-evident. Indeed, the existence of objects must always be verified and the modeler must ensure that the objects are retrieved as intended.

3. Domain modeling using existence dependency

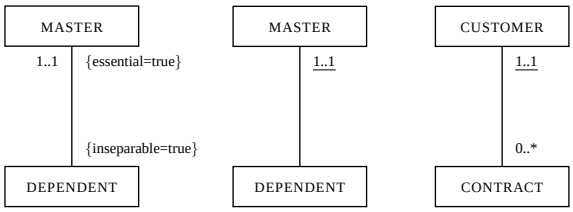
Throughout the following sections we will propose a domain modeling technique in which object (type) relations express existence dependency. In the first sub-section, existence dependency is defined and a graphical representation is proposed. Secondly, we will illustrate that it is possible to create domain models in which *all* object (type) relationships express existence dependency. The resulting existence dependency graphs (EDG) will subsequently be used in the modified version of stateless process modeling and enactment.

3.1. Existence dependency

An object type D is existence dependent on an object type M if each occurrence of D is associated with minimum one, maximum one and always the same occurrence of M [11]. If D is called the dependent object type or simply 'dependent' and M the master object type or simply 'master', existence dependency implies that the life of a dependent object is embedded in the life of the master object. Notice that no restrictions are put on the cardinality on the dependent side of the relation: it has a lower bound of zero or one and an upper bound of one or many.

As an example, consider the relation between CONTRACT and CUSTOMER in figure 2(c) (the notation is explained hereafter), which states that a contract belongs to one customer. The life span of a contract is always embedded in the life span of the customer. Indeed, we cannot have a contract for a customer if the customer does not exist. And the lifecycle of the customer cannot end as long as the lifecycle of the contract is not ended. In addition, a contract always refers to one and the same customer for the whole time of its existence. Hence CONTRACT is existence dependent on CUSTOMER. In this example, a customer can have multiple contracts at the same time, which results in a cardinality of zero-to-many at the side of CONTRACT.

Despite the fact that UML is the de facto standard for, among others, domain modeling, it provides no support for



(a) UML syntax (b) Simplified notation (c) Example

Figure 2. Graphical notation for existence dependency

expressing existence dependency. Even for modeling part-whole relations, the standard is quite imprecise in dealing with *inseparable* parts (i.e. parts that are dependent on the same whole) and *essential* parts (i.e. the whole cannot exist without having a particular object as part). Therefore, Guizzardi et al. [3] propose to add the Boolean-valued 'inseparable' and 'essential' tags for the part association end. Returning to existence dependency, it is clear that dependents are inseparable (they are dependent on the same master) and masters are essential (a dependent cannot exist without a master). Therefore, the 'inseparable' and 'essential' tags should be added to an UML association to express existence dependency, as in figure 2(a). However, since *all* relationships in this paper will express existence dependency, a simpler representation can be used: as in figure 2(b), the master association end has an underlined one-to-one cardinality (1..1) and masters are always drawn above dependents. As such, a specific notation for existence dependency is obtained while maximally adhering to the UML notation.

3.2. Existence dependency graph

Requiring all relationships to express existence dependency allows organizing all the object types of the domain model in an (acyclic) existence dependency graph (EDG). Figure 3 shows an EDG of the insurance domain that was previously modeled using UML (figure 1). Several relations in the UML model did already express existence dependency and are therefore taken over as such, e.g. master-dependent relationships EXPERT-EXPERT'S REPORT, ACCOUNT-INDEMNITY PAYMENT, etc. Furthermore, Snoeck and Dedene [12] show that non-existence dependent associations can be transformed into a object type which is existence dependent of all the object types involved in the association, such that it is actually possible to have all relations express existence dependency. As such, the association is objectified into a class instead of using the association class construct.

Consider the association between CONTRACT and INSURABLE OBJECT in figure 1, which does not express ex-

istence dependency: the life of a contract is not fully contained in the life of the insurable object and neither the other way around. Therefore, the relationship between CONTRACT and INSURABLE OBJECT must be instantiated to a new object type that we call COVERING – see figure 3. This new COVERING object type is existence dependent on both CONTRACT and INSURABLE OBJECT: a covering always refers to one and the same contract and to one and the same insurable object during its whole life. The indicated cardinalities show that one insurable object can be insured in one or more coverings and that a contract can cover multiple insurable objects.

Since all relationships express existence dependency, each master of a dependent object is uniquely navigable. Moreover, the existence of the dependent object implies the existence of its master objects, and these master objects cannot be deleted nor replaced during the lifetime of the dependent object. Obviously, the navigation to master objects can also be carried out recursively, so that different, unique paths can be constructed from an object by navigating from dependent to master in the EDG (upwards).

Navigation can also be carried out in the opposite direction (downwards), i.e. from an object to its dependents. Now it holds that a dependent object can only be created if each of its master objects are existing. Therefore, all object types can be assigned to an existence dependency level as shown in figure 3: all object types without masters belong to level L_1, object types depending on object types in level L_1 belong to level L_2, etc. As such, objects in level L_j cannot be created before objects in level L_i, where $j > 1$ and $i < j$, are created.

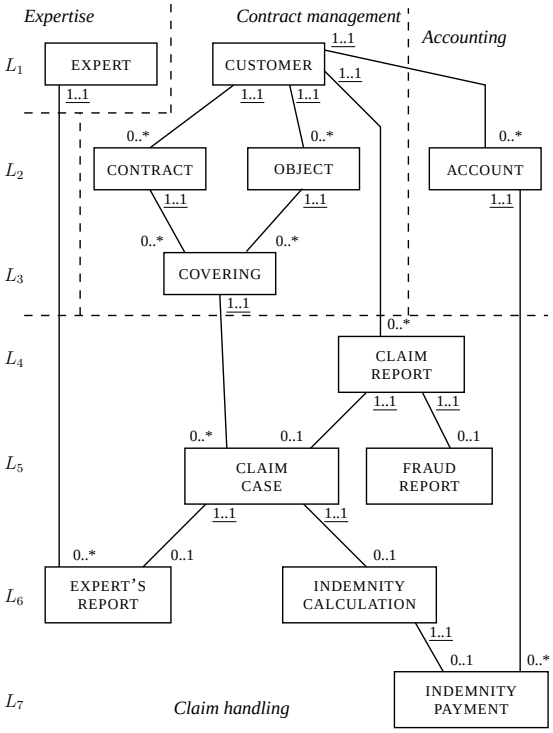


Figure 3. Partial EDG for insurance domain model (object clusters in italic)

4. Revised stateless process enactment

In section 2, we argued that the use of classical UML-based or ER-based domain models prevents comprehensible formulation and efficient verification of the pre- and post-conditions. It was shown that the root causes of these problems are the uncertainty about object existence, the many-to-many relationships and the possibility of re-assigning objects in an association. However, these issues can adequately be disposed of when the business object types are organized according to an EDG. Firstly, the existence of an object implies the existence of objects in master levels that can be reached. For example in figure 3, the existence of an INDEMNITY PAYMENT object implies the existence of a unique master object ACCOUNT, which can be used in the activity 'Pay Indemnity'. Secondly, an EDG does not contain many-to-many relations since the cardinality of the master end in an existence dependency relationship is restricted to one. For example, the many-to-many relationship between CONTRACT and OBJECT in figure 1 is transformed to a COVERING object type in figure 3 that connects a single insurable object and a single contract. Finally, existence dependency entails that a dependent object is associated with one *and the same* master object during the whole life of the dependent.

In what follows, we elaborate on stateless process enactment using an EDG-based domain model and the impact on performance and comprehensibility of the pre- and post-conditions.

Before a process is instantiated for the first time, some kind of preprocessing step must be executed in order to separate the existing from the non-existing objects: all objects in the EDG that nowhere occur as 'existing' in the post-conditions of the activities, must be collected. These objects are not created in any activity and therefore assumed to be already existing prior to the execution of the process. Referring to figure 3, it is logical that the objects in the accounting, contract management and expertise clusters are already existing before the claim handling process starts. These objects are referred to as *necessarily existing objects*.

During process execution, the workflow engine keeps track of references to the most dependent objects (i.e. objects that do not have dependents) of the set of existing objects. We refer to this set of object references as D_{EO}. Initially, D_{EO} consists of the most dependent objects of the necessarily existing objects. This means that in figure 3,

D_{EO} consists of ACCOUNT, COVERING and EXPERT references before the process starts. It is not necessary to maintain references to CUSTOMER, CONTRACT and INSURABLE OBJECT, since these objects can be reached from COVERING. As such, references to only three object types instead of six must be maintained.

Given the previously introduced rules about existence dependency, an object can now be created as soon as its master objects are existent. Moreover, as soon as a particular object is created during an activity, a reference to that object is added to D_{EO} (since the object becomes a most dependent object of the set of existing objects) and the master objects of that object are removed from D_{EO} (since these masters are reachable from the newly created object).

Once again referring to figure 3, only CLAIM REPORT objects can initially be created according to the EDG. The set D_{EO} now consists of ACCOUNT, COVERING, EXPERT and CLAIM REPORT references and at this point, both CONTRACT ASSIGNMENTS and FRAUD REPORTS can be created. If a CONTRACT ASSIGNMENT is actually created, that object is added to D_{EO} whereas COVERING and CLAIM REPORT (the masters of CONTRACT ASSIGNMENT) references can be removed from D_{EO}.

It should be noted that pre- and post-conditions allow more facts to be verified besides the existence of objects. Obviously, the values of object attributes may also be verified in the pre- and post-conditions, or more generally, whether an attribute is set or not. However, these conditions do not change the set of existing objects (hence neither D_{EO}) and are therefore more straightforward to deal with. Additionally, the non-existence of objects can also be verified, which corresponds to object deletion when stated as a post-condition. In this case, it is important to note that an object can only be deleted when it has no existing dependent objects. Finally, a deleted object must be removed from D_{EO} whereas its master objects must be added to the set.

4.1. Analysis and discussion

By following the above approach, D_{EO} always represents the minimal set of object references from which all other existing objects can be derived. As such, the amount of stored object references is drastically reduced and above this, the checking of pre- and post-conditions can considerably be improved: once again it follows from the definition of existence dependency that an object O does not exist iff O (potentially transitively) dependents on an object in D_{EO}. Similarly, an object O exists iff O is a (potentially transitive) master object of an object in D_{EO}. As such, the existence of objects and therefore the pre- and post-conditions can be checked in a more efficient way.

Besides improving performance, the proposed approach also makes the specified conditions more comprehensible. Indeed, in an EDG, a unique path exists from each existing object to each of its master objects. It was also shown that the navigation to master objects can be carried out recursively, so that each existing object can unambiguously be reached from the objects in D_{EO}. For example, if the COVERING object exists in figure 3, conditions can refer to that object and its master objects, i.e. the contract that describes the covering (covering.contract) and the object that is covered (covering.insurable object). Moreover, in the conditions it is possible to distinguish between the customer that signed the contract (covering.contract.customer) and the customer that owns the insurable object (covering.insurable object.customer). The EDG clearly shows to the modeler the different objects that can be reached via different paths.

To conclude this discussion, we place emphasis on the remarkable duality between process modeling and data modeling when an EDG is used as domain model in the context of stateless process enactment. Indeed, the objects in D_{EO} form some kind of 'wavefront' above which all objects are existing and under which all objects are not (yet) existing. As explained in the previous section, these objects in D_{EO} can be seen as future master objects of objects that are currently situated underneath the wavefront. As such, the sum of the number of dependent objects of each object in D_{EO} gives an indication of the process flexibility at a particular point of execution. For example according to the domain model in figure 3, both a contract assignment and/or a fraud report can be created (in any order) as soon as a claim report exists. Obviously, this 'metric' only takes into account the flexibility that emanates from activities that create objects. However, since the existing objects are known at each moment, it is also possible to assess the flexibility that is caused by other activities that update attributes or delete objects.

5. Related work

Business process modeling (BPM) is mostly considered from an *activity-centric* perspective, which focuses on the actions to be taken and the control flow between these actions. On the other hand, *information-centric* (or *artifact-centric*) process models focus on the business objects that are acted upon [6, 2, 8, 10]. In these works, a business process is generally defined in terms of a set of (interacting) object life cycles, whereas in our approach, the process model is related with a set of business objects by means of activity pre- and post-conditions. As such, our approach can also be considered as information-centric.

Most works on information-centric BPM emphasize on the improvements on activity-centric BPM. For example, it is argued that information-centric process models are more

flexible [2], more easily managed and agreed on [8] and they facilitate integration of people, processes, information and applications [10]. This paper addresses some issues that may occur when information-centric models are designed and used for process enactment. More specifically, we proposed a specific data modeling technique that improves comprehensibility of the process models and performance during enactment.

Stateless process enactment builds on one of the key principles of the case handling paradigm, which states that an activity is completed if the associated mandatory data objects are entered [13]. It differs from our approach since (a) only post-conditions of an activity can be expressed in terms of mandatory data objects that must exist and (b) a data object is the single fine-grained construct to express post-conditions. The pre- and post-conditions in our approach express facts about business objects that are structured according to a particular domain model.

The link between process models and data models is discussed in multiple works. For example, De Backer et al. [1] propose a technique for verifying compliance between activity-centric process models and a set of object life cycles. Other authors guarantee compliance by automatically generating process models from object life cycles [7] or the other way around [9]. To the latter category also belongs the method that is proposed by Kumaran et al [6]. The authors state that object A dominates object B if, for each activity, the potential occurrence of B as input (output) always implies the occurrence of A as input (output). As such, state information of the dominated object B is not needed for deriving the state of the process and can therefore be discarded. This approach clearly resembles ours since in both cases, it searches for the minimal set of objects that is needed to reconstruct process state. However, our approach is based on the specification of pre- and post-conditions instead of input/output and can therefore deal with incomplete models.

6. Conclusions and future work

In this paper we elaborated on domain modeling for stateless process enactment. More specifically, to enable stateless process enactment, activities have pre- and post-conditions and these conditions refer to business object that are organized in a particular domain model. We showed that traditional domain modeling techniques (e.g. UML or ER) yield models in which the coherence between objects is weak: the existence of objects is never guaranteed, the many-to-many relationships prevent unique navigation and it is possible to re-assign objects in an association. Because of these reasons, all object references must be maintained and all pre- and post-conditions must be verified, which clearly deteriorates performance. For the same reasons, the formulation of clear and uniform (i.e. comprehensible) rules is not self-evident.

In order to alleviate these issues, we presented an alternative domain modeling technique that is based on the notion of existence dependency. A dependent object is existence dependent on a master object if the dependent is associated with one and the same master during the lifetime of the dependent. If all relations between objects express existence dependency, an existence dependency graph (EDG) as domain model is obtained. From these definitions it follows that an EDG does not contain many-to-many relations, that the existence of object implies the existence of its (transitive) masters and that a master can not be re-assigned. As such, it was shown that the number of maintained objects can be decreased and that navigation in an EDG caters for comprehensible pre- and post-conditions.

As part of future work, we will elaborate on the incorporation of the modeling of object life cycles in our approach. A life cycle defines the permissible object states, which can also be used for deriving process state. We will investigate the trade-off between the addition of a new object and the addition of a new state to an existing object. Finally, we will create a prototype in order to demonstrate the feasibility of stateless process enactment.

7. Acknowledgments

This work was funded by the KBC-HUBrussel-K.U.Leuven research chair on Service and Component Based Development sponsored by KBC Global Services.

References

[1] M. D. Backer, M. Snoeck, G. Monsieur, W. Lemahieu, and G. Dedene. A scenario-based verification technique to assess the compatibility of collaborative business processes. *Data & knowledge engineering*, 2009.

[2] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304, Brisbane, Australia, 2007. Springer-Verlag.

[3] G. Guizzardi, H. Herre, and G. Wagner. Towards ontological foundations for uml conceptual models. In *Proceedings of the 1st International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2002)*, Lecture Notes in Computer Science, pages 1100–1117, London, UK, 2002. Springer-Verlag.

[4] R. Haesen, L. De Rore, S. Goedertier, M. Snoeck, W. Lemahieu, and S. Poelmans. Stateless process enactment. In *Preliminary Proceedings of the 14th Conferences on Pattern Languages of Programming (PLoP 2007)*, Illinois, USA, September 2008.

[5] R. Haesen, S. Goedertier, K. Van de Cappelle, W. Lemahieu, M. Snoeck, and S. Poelmans. A phased deployment of a workflow infrastructure in the enterprise architecture. In *Proceedings of The International Workshop on Collaborative Business Processes (CBP 2007) at BPM 2007*, volume 4928 of *Lecture Notes in Computer Science*, pages 268–278, Brisbane, Australia, 2008. Springer Verlag.

[6] S. Kumaran, R. Liu, and F. Y. Wu. On the duality of information-centric and activity-centric models of business processes. In *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, pages 32–47, 2008.

[7] J. M. Küster, K. Ryndina, and H. Gall. Generation of business process models for object life cycle compliance. In *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, number 4714 in Lecture Notes in Computer Science, pages 165–181, Brisbane, Australia, 2007. Springer-Verlag.

[8] R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, volume 4495 of *Lecture Notes in Computer Science*, pages 324–339, Trondheim, Norway, 2007. Springer-Verlag.

[9] D. Müller, M. U. Reichert, and J. Herbst. Flexibility of data-driven process structures. In *International Workshops at BPM 2006*, volume 4103 of *Lecture Notes in Computer Science*, pages 179–190. Springer-Verlag, 2006.

[10] P. Nandi and S. Kumaran. Adaptive business objects - a new component model for business integration. In *Proceedings of the 7th International Conference on Enterprise Information Systems (ICEIS 2005)*, pages 179–188, Miami, USA, 2005.

[11] M. Snoeck and G. Dedene. Existence dependency: The key to semantic integrity between structural and behavioral aspects of object types. *IEEE Transactions of Software Engineering*, 24(4):233–251, 1998.

[12] M. Snoeck and G. Dedene. Core modelling concepts in object-oriented conceptual modelling. In *Proceedings of the 38th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS '01)*, pages 170–179. IEEE Computer Society, 2001.

[13] W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.

[14] WFMC. The workflow reference model. Technical Report WFMC-TC-1003, Workflow Management Coalition, 1995.